Spanner: Google's Globally Distributed Database

(presented by Philipp Moritz)
Why is this workload interesting?

- SQL $\rightarrow$ NoSQL $\rightarrow$ NewSQL
- Large scale transactional databases
- Eventual consistency is not good enough (?):
  - Managing global money/warehouses/resources
  - Auctions, especially Google's advertisement platform
  - Social networks, Twitter
  - MapReduce over a globally changing dataset
- We need external consistency:
  \[ T(e1(\text{commit})) < T(e2(\text{start})) \rightarrow s1 < s2 \]
Concepts

• Main idea:
  – Get externally consistent view of globally distributed database
  – Spanner = BigTable with timestamps + Paxos + TrueTime
• Details:
  – Globally distributed for locality and fault-tolerance
  – Automatic load balancing between datacenters
  – Semirelational + SQL like query language (cf. Dremel)
  – Versioning
  – Full control over
    • How far data is from user (read latency)
    • How far replicas are from each other (write latency)
    • How many replicas (durability, availability, throughput)
Paxos in a Nutshell

- Algorithm for finding consensus in a distributed system

```
<table>
<thead>
<tr>
<th>Client</th>
<th>Proposer</th>
<th>Acceptor</th>
<th>Learner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X-------&gt;</td>
<td></td>
<td></td>
<td>Request</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Prepare(1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Promise(1,{Va,Vb,Vc})</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Accept!(1,Vn)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Accepted(1,Vn)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X---------&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Response</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X--X</td>
</tr>
</tbody>
</table>
```

TrueTime

- Goal: Provide globally synchronized time with sharp error bounds
- Do not trust synchronization via NTP
- With GPS and “commodity” atomic clocks, Google created their own time standard
- TrueTime API:
  - TT.now(): Interval [earliest, latest]
  - TT.after(t): true if t has definitely passed
  - TT.before(t): true if t has definitely not arrived
- Spanner implements algorithms to make sure these guarantees are respected by the machines (non-conformists are evicted)
- Time accuracy on the order of 10ms
Spanservers

The diagram illustrates a distributed system architecture for Spanservers. It consists of multiple replicas and tablets spread across different data centers. Each data center contains tablets and replicas, which are connected through Paxos protocol. The system is managed by a leader and a lock manager to ensure consistency and reliability.
Interplay of Paxos and TrueTime

- Guarantee externally consistent transactions

\[
\begin{align*}
    s_1 & < t_{abs}(e_1^{commit}) \\
    t_{abs}(e_1^{commit}) & < t_{abs}(e_2^{start}) \\
    t_{abs}(e_2^{start}) & \leq t_{abs}(e_2^{server}) \\
    t_{abs}(e_2^{server}) & \leq s_2 \\
    s_1 & < s_2
\end{align*}
\]

(commit wait)  
(assumption)  
(causality)  
(start)  
(transitivity)
Table 3: Operation microbenchmarks. Mean and standard deviation over 10 runs. 1D means one replica with commit wait disabled.

<table>
<thead>
<tr>
<th>participants</th>
<th>latency (ms)</th>
<th>99th percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17.0 ± 1.4</td>
<td>75.0 ± 34.9</td>
</tr>
<tr>
<td>2</td>
<td>24.5 ± 2.5</td>
<td>87.6 ± 35.9</td>
</tr>
<tr>
<td>5</td>
<td>31.5 ± 6.2</td>
<td>104.5 ± 52.2</td>
</tr>
<tr>
<td>10</td>
<td>30.0 ± 3.7</td>
<td>95.6 ± 25.4</td>
</tr>
<tr>
<td>25</td>
<td>35.5 ± 5.6</td>
<td>100.4 ± 42.7</td>
</tr>
<tr>
<td>50</td>
<td>42.7 ± 4.1</td>
<td>93.7 ± 22.9</td>
</tr>
<tr>
<td>100</td>
<td>71.4 ± 7.6</td>
<td>131.2 ± 17.6</td>
</tr>
<tr>
<td>200</td>
<td>150.5 ± 11.0</td>
<td>320.3 ± 35.1</td>
</tr>
</tbody>
</table>

Table 4: Two-phase commit scalability. Mean and standard deviations over 10 runs.
Discussion

- Tradeoff: Complexity of the System vs. Importance of Guarantees
- Is eventual consistency good enough if the operations we care about are fast enough?
- If not: Can we isolate a small subset of data for which we care about consistency and store it on a single server?
- Open Source implementation of similar ideas: https://github.com/cockroachdb/cockroach